JOURNAL OF COMPUTER AND SYSTEM SCIENCES 7, 389–403 (1973)

Maze Recognizing Automata and Nondeterministic Tape Complexity*

WALTER J. SAVITCH

Department of Applied Physics and Information Science, University of California at San Diego La Jolla, California 92037

Received May 19, 1972

A new device called a maze recognizing automaton is introduced. The following two statements are shown to be equivalent. (i) There is a maze recognizing automaton that accepts precisely the threadable mazes. (ii) Every nondeterministic $L(n)$-tape bounded Turing machine can be simulated by a deterministic $L(n)$-tape bounded Turing machine, provided $L(n) \geqslant \log_2 n$.

INTRODUCTION

A new device called a maze-recognizing automaton is introduced. This device is a type of finite-state machine that crawls through mazes. It is shown that maze-recognizing automata and $\log n$ tape-bounded Turing machines are equally potent for recognizing threadable mazes. Whether or not maze-recognizing automata can recognize threadable mazes is an open question. This problem is related to the problem of simulating nondeterministic algorithms by deterministic algorithms. In particular, it is shown that if there is a maze-recognizing automaton which can recognize threadable mazes, then every context-sensitive language is accepted by a deterministic linear-bound automaton. This paper is an extension of the work done in [2], and a familiarity with that paper will be helpful to the reader.

By *Turing machine* we will mean a Turing machine with a separate read-only input tape and finitely many read-write storage tapes. We assume that the input is delimited by end markers. A Turing machine Z, (deterministic or nondeterministic) is said to accept the set A within storage $L(n)$ provided that Z accepts exactly the set A and furthermore: for each w in A, there is at least one possible computation of Z which accepts w and in which each storage tape head scans at most $L(|\, w \,|)$ tape squares. $|\, w \,|$ is the length of the string w. Detailed definitions of these concepts may be found in [1] and [2]. Any Turing machine which operates in storage $L(n)$

* This research was supported by NSF grant #GJ-27278.

can be modified to operate in storage $\epsilon L(n)$ and still accept the same set of tapes, where ϵ is an arbitrary constant greater than zero. We assume the reader is familiar with such tape reduction techniques (see, e.g., [1]).

Mazes

DEFINITION. A *maze* over Σ (a finite alphabet) is a quadruple, $\mathcal{M} = (X, R, s, G)$, where X is a finite set of strings over Σ (X is the set of rooms), R is a binary relation on X (giving the corridors), s is an element of X (s is the start room), and G is a subset of X (G is the set of goal rooms).

DEFINITION. The maze $\mathcal{M} = (X, R, s, G)$ is *threadable* if there is a sequence $r_1, r_2, \ldots, r_e$ of rooms such that $r_1 = s$ (the start room), r_e is an element of G (the goal rooms), and $R(r_i, r_{i+1})$ holds for $i = 1, 2, \ldots, e - 1$.

DEFINITION. Let], [, * be three new symbols. A *coding* of the maze $\mathcal{M} = (X, R, s, G)$ is a string of the form

$$s[x_1 * y_1^1 * y_2^1 * \cdots y_{n(1)}^1][x_2 * y_1^2 * y_2^2 * \cdots * y_{n(2)}^2]$$
$$\cdots [x_l * y_1^l * \cdots y_{n(l)}^l]\, u_1 * u_2 * \cdots u_g ,$$

where s is the start room of $\mathcal{M}$; $x_1, x_2, \ldots, x_l$ is an enumeration without repetitions of the rooms in X; for $1 \leqslant i \leqslant l$, $y_1^i, y_2^i, \ldots, y_{n(i)}^i$ is an enumeration without repetitions of all y in X such that $R(x_i, y)$ holds; and $u_1, u_2, \ldots, u_g$ is an enumeration without repetitions of the rooms in G.

NOTATION. M_Σ denotes the set of all *codings of threadable mazes* over Σ.

DEFINITION. A maze $\mathcal{M} = (X, R, s, G)$ is called a *2-branch maze* (abbreviated *2-maze*) if, from every room, there are at most two rooms accessible in one step. That is: For each x in X there are at most two distinct y in X such that $R(x, y)$ holds.

NOTATION. M_Σ^2 denotes the set of all *codings of threadable 2-mazes* over Σ.

Maze Recognizing Automata

Informally a maze-recognizing automaton (MRA) $\mathcal{A}$, is a finite-state machine provided with finitely many distinguished pebbles. It operates on 2-mazes such that from each room of the maze there are exactly two corridors leading out to other

rooms. The corridors are labeled 1 and 2. The rooms of the maze are numbered 1 through N. At any point in time $\mathcal{A}$ is in some room, in some state, with some pebbles in its possession, and the remaining pebbles are distributed in some way among the rooms. $\mathcal{A}$ can distinguish its state, whether it is in a start room, goal room, or neither and can tell which pebbles are in the room as well as which pebbles are in its possession. On the basis of this information it will, in a deterministic way, change state, drop or pick up pebbles, and finally either move through corridor 1 or 2 to a new room or move to the room with the next highest number (i.e., move from room m to room number $m + 1$). For the last type of move, 1 is considered to be the successor of N (the highest numbered room). Certain states of $\mathcal{A}$ are distinguished and called accepting states; one state is distinguished and called the initial state. $\mathcal{A}$ accepts the maze $\mathcal{M}$ if $\mathcal{A}$ ever reaches an accepting state in the computation starting from the initial configuration in which $\mathcal{A}$ is in the start room of $\mathcal{M}$, in the initial state with all its pebbles in its possession.

All of the results appear to depend critically on the ability of an MRA to move from room to room in numerical order. This type of move allows an MRA to get from any room to any other room without having to actually thread the maze. Some such device is necessary in order to allow the MRA to "look at" the maze. As we shall see, this ability to cycle through the rooms also allows the MRA to use the rooms as a type of counter and so obtain some general-purpose storage.

DEFINITION. A *maze recognizing automaton* (*MRA*), $\mathcal{A}$, is a quintuple (Q, P, δ, q_0, F) where:

(i) Q and P are finite sets (of states and pebbles respectively);

(ii) δ is a partial function from $Q \times 2^P \times 2^P \times \{0, S, G\}$ to $Q \times 2^P \times 2^P \times \{1, 2, +\}$ such that if $\delta(q, A, B, U) = (q', A', B', V)$, then $A' \cap B'$ is empty and $A' \cup B' = A \cup B$ (2^P, as usual, denotes the collection of all subsets of P);

(iii) q_0 is an element of Q (q_0 is the initial state);

(iv) F is a subset of Q (F is the set of accepting states).

Intuitively (ii) means the following. Suppose $\mathcal{A}$ is in state q and that A is the set of pebbles in its possession. Suppose further that $\mathcal{A}$ is in room number m, B is the set of pebbles in room m, and room m is either the start room, goal room or neither depending on whether U is S, G or 0 respectively. Under these conditions, $\mathcal{A}$ will redistribute pebbles $A \cup B$ so that A' becomes the set of pebbles in $\mathcal{A}$'s possession and B' becomes the set of pebbles in room m. $\mathcal{A}$ will then change its internal state to q'. Finally, $\mathcal{A}$ will move to the room at the end of corridor V if $V = 1$ or 2, or to room $m + 1$ if V is $+$. If m is the largest number for any room and V is $+$, then $\mathcal{A}$ moves to room number 1 (since in this case there is no room number $m + 1$). All this constitutes a single atomic move of $\mathcal{A}$. MRA's operate on structures called 2-branch numbered mazes.

DEFINITION. A *2-branch numbered maze* (abbreviated *numbered 2-maze*) is a quadruple $\mathcal{M} = (X, \sigma, s, G)$ where:

(i) $X = \{1, 2, \ldots, N\}$ for some integer N (X is the set of rooms);

(ii) σ is a function from $X \times \{1, 2\}$ to X (giving the corridors);

(iii) s is an element of X (s is the start room);

(iv) G is a subset of X (G is the set of goal rooms).

TERMINOLOGY. Let the notation be as in the previous definition. Define the binary relation R_σ on X by: $R_\sigma(x, y)$ holds if and only if $\sigma(x, i) = y$ for $i = 1$ or $i = 2$. Identify the numbers in X with their m-ary representation. (X, R_σ, s, G) then satisfies the definition of 2-maze given in the last section. So we can carry over concepts about mazes defined in the last section. In particular, to say $\mathcal{M}$ is *threadable* means that (X, R_σ, s, G) is threadable and by a *coding* of $\mathcal{M}$ we will mean a coding of (X, R_σ, s, G) over the alphabet $\{0, 1, 2, \ldots, m - 1\}$.

The concept of a numbered 2-maze differs from the concept of a 2-maze given previously in that rooms in a numbered 2-maze always have exactly two corridors leading out of them. Furthermore, these corridors are distinguished; one is labeled 1 and the other is labeled 2. (Of course, the two corridors could go to the same room.) Also, a numbered 2-maze has its rooms numbered sequentially.

NOTATION. Let $\mathcal{M} = (X, \sigma, s, G)$ be a numbered 2-maze with $X = \{1, 2, \ldots, N\}$. We extend σ to a function from $X \times \{1, 2, +\}$ to X by setting $\sigma(m, +) = m + 1$, if $m \neq N$ and $\sigma(N, +) = 1$. For any room n, $\sigma(n, +)$ is called the *successor* of n.

DEFINITION. An *instantaneous description* (ID) of the MRA $\mathcal{A} = (Q, P, \delta, q_0, F)$ on the numbered 2-maze $\mathcal{M} = (X, \sigma, s, G)$ is a triple (n, q, μ) where:

(i) n is in X (n is the number of the room $\mathcal{A}$ is in);

(ii) q is in Q (q is the internal state of $\mathcal{A}$);

(iii) μ is a function from $X \cup \{\mathcal{A}\}$ to 2^P such that $\bigcup_x \mu(x) = P$, where x ranges over $X \cup \{\mathcal{A}\}$, and $\mu(x) \cap \mu(y)$ is empty if $x \neq y$.

μ gives the distribution of pebbles. $\mu(x)$ is the set of pebbles in room x and $\mu(\mathcal{A})$ is the set of pebbles held by $\mathcal{A}$.

DEFINITION. For any two ID's (n, q, μ) and (n', q', μ') of the MRA $\mathcal{A}$ on the numbered 2-maze $\mathcal{M}$ we write $(n, q, \mu) \vdash_{\mathcal{A},\mathcal{M}} (n', q', \mu')$ provided $\delta(q, \mu(\mathcal{A}), \mu(n), U) = (q', \mu'(\mathcal{A}), \mu'(n), V)$ and $n' = \sigma(n, V)$, where U is S, G, or O respectively depending on whether n is a start room, goal room or neither. And provided $\mu(x) = \mu'(x)$, for all rooms x other than n. (σ and δ are as in the previous definition.)

Intuitively $(n, q, \mu) \vdash_{\mathcal{A},\mathcal{M}} (n', q', \mu')$ means that if $\mathcal{A}$ is in configuration (n, q, μ) on $\mathcal{M}$, then in one move $\mathcal{A}$ will go to configuration (n', q', μ'). $\vdash^*_{\mathcal{A},\mathcal{M}}$ will denote the reflexive, transitive closure of $\vdash_{\mathcal{A},\mathcal{M}}$. That is, $(n, q, \mu) \vdash^*_{\mathcal{A},\mathcal{M}} (n', q', \mu')$ means we can get from (n, q, μ) to (n', q', μ') by a finite number of applications of $\vdash_{\mathcal{A},\mathcal{M}}$.

DEFINITION. The MRA $\mathcal{A}$ *accepts* the numbered 2-maze $\mathcal{M}$ provided that for some accepting state q, some room n of $\mathcal{M}$ and some μ, $(s, q_0, \mu_0) \vdash^*_{\mathcal{A},\mathcal{M}} (n, q, \mu)$ where s is the start room of $\mathcal{M}$, q_0 is the initial state of $\mathcal{A}$ and μ_0 is defined by: $\mu_0(\mathcal{A}) = P$ (the set of all pebbles) and $\mu_0(x)$ is the empty set, for all x in X.

MAZE RECOGNIZING AUTOMATA AND TURING MACHINES

We are now ready to show that constructing an MRA which accepts exactly the threadable 2-mazes is equivalent to constructing a deterministic $\log_2 n$, tape-bounded Turing machine which recognizes codings of threadable mazes. That is, we can now prove the following theorem.

THEOREM 1. *Let Σ be a finite alphabet with at least two elements. Then the following statements are equivalent:*

(1) *$M_\Sigma{}^2$ is accepted by some deterministic Turing machine within storage* $\log_2 n$; *and*

(2) *there is a* MRA, *$\mathcal{A}$, such that for any numbered* 2-*maze $\mathcal{M}$, $\mathcal{A}$ accepts $\mathcal{M}$ if and only if $\mathcal{M}$ is threadable.*

Proof. Suppose there is an MRA $\mathcal{A}$, which accepts exactly the threadable numbered 2-mazes. It is routine to construct a Turing machine Z to recognize $M_\Sigma{}^2$ by mimicking $\mathcal{A}$. More specifically, Z operates as follows. Z can easily be designed to reject any string of symbols that is not a coding of a 2-maze. So we may assume Z receives input strings of the form

$$(*) \quad x_{i_0}[x_1 * y_1{}^1 * y_2{}^1][x_2 * y_1{}^2 * y_2{}^2] \cdots [x_N * y_1{}^N * y_2{}^N]\, x_{i_1} * x_{i_2} * \cdots * x_{i_g}$$

where $x_1, x_2, \ldots, x_N \in \Sigma^*$ and the x_j are distinct. (*) codes a 2-maze which is isomorphic to the numbered 2-maze $\mathcal{M} = (X, \sigma, s, G)$, where $X = \{1, 2, \ldots, N\}$, $s = i_0$, $G = \{i_1, i_2, \ldots, i_g\}$ and σ is defined by $\sigma(i, 1) = \#y_1{}^i$ and $\sigma(i, 2) = \#y_2{}^i$ for $i = 1, 2, \ldots, N$. $\#y_1{}^i$ is the unique natural number j such that $y_1{}^i = x_j$. In a 2-maze there are at most two rooms accessible from any x_i. However, there may not be exactly two rooms accessible from x_i. In this case, make the extra corridors go back into room i. In other words, if some $y_k{}^i$ does not occur in (*), set $\sigma(i, k) = i$.

Z must determine if (*) codes a threadable maze. That is, Z must determine if $\mathcal{M}$ is threadable. Z does this by simulating $\mathcal{A}$ operating on $\mathcal{M}$. In order to perform

this simulation, Z must be able to store any ID of $\mathcal{A}$ on $\mathcal{M}$ and must be able to update this ID according to what $\mathcal{A}$ would do. In order to record an ID of $\mathcal{A}$ all Z needs to record is the number of the room $\mathcal{A}$ is in, the number of the room each pebble is in and the state of the finite control of $\mathcal{A}$. This it can easily store in $c \log_2 N$ squares of tape, for some constant c. The transition function of $\mathcal{A}$ is stored in the finite control of Z. So that in order to update an ID of $\mathcal{A}$, Z need only be able to compute $\sigma(i, 1) = \#y_1^i$, $\sigma(i, 2) = \#y_2^i$ and $\sigma(i, +) = i + 1$, given i. To accomplish this, Z first locates x_i in (*). This requires $\log_2 i$ storage. Z then computes $\#y_1^i$ as follows. Z compares y_1^i to each x_j. Z does this symbol by symbol. To do this Z need only remember the positions of the symbols being compared. So the total storage needed is proportional to $\log_2 n$, where n is the length of (*). Having located that x_j such that $x_j = y_1^i$, Z can easily compute j in $\log_2 j$ storage. $\#y_2^i$ is computed in the same way. So the total storage need is bounded by $c \log_2 n$, where n is the length of the input (*) and c is a constant depending only on Z. Thus (2) implies (1), so it remains to show that (1) implies (2).

Suppose (1) is true and Z is the deterministic Turing machine which recognizes $M_\Sigma{}^2$ within storage $\log_2 n$. We will construct an MRA, $\mathcal{A}$, which accepts precisely the threadable mazes. $\mathcal{A}$ will operate by mimicking Z. As we shall see, $\mathcal{A}$ can use its pebbles and the rooms of the maze as a counter to count up to about n. This is the trick which allows it to simulate $\log_2 n$ tape storage. The details of $\mathcal{A}$'s operation are as follows.

Suppose $\mathcal{A}$ is operating on the 2-maze $\mathcal{M} = (X, \sigma, s, G)$ where $X = \{1, 2, \ldots, N\}$. $s = 1$ and $G = \{u_1, u_2, \ldots, u_g\}$ where $u_1 < u_2 < \cdots < u_g$. Since $\mathcal{A}$ cannot recognize what the actual number of a room is but can merely detect the cyclic ordering $1 < 2 < 3 < \cdots < N < 1$, there is no loss of generality in assuming the start room is room number 1. We may identify Σ^* with the base m numerals, where m is the number of elements in Σ. So, with this identification, the rooms $1, 2, \ldots, N$ are all elements of Σ^*. $\mathcal{A}$ will operate by simulating Z operating on the following coding of $\mathcal{M}$.

$$(**) \qquad 1[1 * \sigma(1, 1) * \sigma(1, 2)][2 * \sigma(2, 1) * \sigma(2, 2)] \cdots [N * \sigma(N, 1) * \sigma(N, 2)]\, u_1 * u_2 * \cdots u_g$$

(If $\sigma(i, 1) = \sigma(i, 2)$, then $* \sigma(i, 2)$ should be omitted from (**). Otherwise (**) does not technically satisfy the definition of a *coding* of a maze. To simplify the notation, we will assume the maze is coded as in (**), even if some $\sigma(i, 1) = \sigma(i, 2)$. Z can be modified to ignore $\sigma(i, 2)$ whenever $\sigma(i, 2) = \sigma(i, 1)$. So there is no loss of generality in this assumption. In any event, the problem is a purely notational one; the proof is the same for any reasonable convention on codings.)

Suppose Z has h storage tapes and k storage tape symbols. Since $\log_k$ (length of (**)) is bounded by a constant multiple of $\log_k N$, there is no loss of generality in assuming

TABLE I

	ID of Z	ID of $\mathcal{A}$
	INPUT HEAD	
0.	Input head reading left or right end marker	Remembered in finite control
1.	Input head reading initial 1	Remembered in finite state control
2.	Input head reading in $[m*\sigma(m, 1)*\sigma(m, 2)]$	Pebble in room number m (If input is not in a block of this form then $\mathcal{A}$ picks up this pebble.)
3.	Which one of $[m, *\sigma(m, 1)$ or $*\sigma(m, 2)]$ input head is scanning	Remembered in finite state control
4.	Input head scanning i-th digit from the right of $[m, *\sigma(m, 1)$ or $*\sigma(m, 2)]$ (whichever is appropriate)	Pebble in room number i
5.	Input head reading in $*u_j$	Pebble in room number u_j (if input is not in a block of this form then $\mathcal{A}$ picks up this pebble.)
6.	Input head scanning i-th digit from the right of $*u_j$	Pebble in room number i
	STORAGE TAPES	
7.	j-th storage tape contains the k-ary representation of m $(j = 1, 2, \ldots, h)$	Pebble in room number m
8.	j-th storage tape head scanning i-th digit from the right of $(m)_k$, where $(m)_k$ is the k-ary representation of m $(j = 1, 2, \ldots, h)$	Pebble in room number i
	FINITE CONTROL	
9.	State of finite control of Z	Remembered in finite state control

that each storage tape of Z is bounded by $\log_k N$. Assume that each storage tape of Z is bounded by $\log_k N$. Then each storage tape holds the k-ary representation of a number which is less than or equal to N, the number of rooms in the maze $\mathcal{M}$.

We first show how an ID of Z is coded into an ID of $\mathcal{A}$. Later we will show how $\mathcal{A}$ can update this ID according to what Z would do. Table I gives the representation of the various aspects of an ID of Z operating on the coding (**) of $\mathcal{M}$ in terms of an ID of $\mathcal{A}$ operating on $\mathcal{M}$ itself. Recall that the various pebbles in an MRA can be distinguished. For each entry in Table I, $\mathcal{A}$ uses a distinct pebble.

Thus any ID of Z can be recorded in the configuration of $\mathcal{A}$. So, in order to show that $\mathcal{A}$ can indeed be designed to simulate Z, we need only show that, with the aid of finitely many additional pebbles, $\mathcal{A}$ can, in finitely many steps, change this record of Z's ID in accordance with what Z would do. We first show that $\mathcal{A}$ can perform three simpler tasks. We then show how $\mathcal{A}$ can change the record of Z's ID by using these tasks as subroutines. The three tasks are the following.

1. *Find Predecessor.* Starting with pebble I in room number m, determine if $m = 1$ and if not, move pebble I to room $m - 1$. (We are still assuming that room number 1 is the start room.)

2. *Recover i-th Digit.* Starting with pebble I in room number m and pebble II in room number i, determine if the length of $(m)_k$ is greater than i and if not, discover the i-th digit (from the right) of $(m)_k$ and store it in the finite control. $(m)_k$ denotes the k-ary representation of m. (k is fixed.)

3. *Change i-th Digit.* Starting with pebble I in room number m and pebble II in room number i, with $i \leqslant$ length $(m)_k$, move pebble I to room m', where m' is that number such that the i-th digit of $(m')_k$ is d and otherwise $(m')_k$ is identical to $(m)_k$ (d and k are fixed). As usual, $(y)_k$ denotes the k-ary representation of the number y.

The following is a program for an MRA to perform task 1. *Find Predecessor.* It requires one additional pebble. Initially pebble I is in room m.

START: Test: Pebble I is in the start room?
Yes: $m = 1$. STOP.
No: Continue.
Place pebble II in room number 1 (i.e. in the start room).

CHECK: Go to the successor of the room containing pebble II.
Test: This room contains pebble I?
Yes: Go to CLEANUP.
No: Continue
Find pebble II and move it to the successor of the room it is in.
Go to CHECK

CLEANUP: Find pebble I and move it to the room containing pebble II.
STOP

We may think of a pebble as a register capable of holding any natural number less than or equal to the number of rooms in the maze. A pebble in room number m is thought of as a register containing the number m. Since task 1 (Find Predecessor) can be performed, it follows that an MRA is capable of incrementing or decrementing any of these registers by one. It can tell if the register is zero. The MRA decrements the register from one (pebble in start room) to zero by picking up the pebble. From this it follows that it can compare two registers to see if they are equal, set one register equal to another, and add or subtract the contents of one register to another so long as the total does not exceed the number of rooms in the maze. The calculating power of an MRA using pebbles as registers may be summarized as follows: An MRA can compute any function computable on a Shepherdson–Sturgis limited register machine [3] by a program in which the contents of any register never exceeds the maximum value of any of the inputs. Implementing tasks 2 and 3 on these registers is routine, and so will not be given here. The techniques involved may be found in [3].

It remains to show how $\mathcal{A}$ uses these tasks to update the coding of the Turing machine ID. Using task 2 (recover i-th digit), $\mathcal{A}$ can immediately recover the symbols scanned by each head of the Turing machine, Z. $\mathcal{A}$ then knows what actions of Z to simulate. $\mathcal{A}$ can then update the coding of each storage tape configuration by a direct application of task 3 (change i-th digit). Finally, $\mathcal{A}$ updates the coding of the input head position. Exactly how this is accomplished depends on what portion of input Z is scanning. The procedures are tabulated below by cases. Pebble numbers refer to Table 1.

Case 1. *Input head reading in* $[m * \sigma(m, 1) * \sigma(m, 2)]$. In this case the finite control remembers which one of the three strings $[m, * \sigma(m, 1)$ or $* \sigma(m, 2)]$ is being scanned and one pebble remembers which digit of that string, viewed as a numeral, is being scanned. As long as the required head movement does not leave the block $[m * \sigma(m, 1) * \sigma(m, 2)]$, a simple change of finite control and movement of one pebble will update the configuration. If the required head movement leaves the block $[m * \sigma(m, 1) * \sigma(m, 2)]$, then $\mathcal{A}$ proceeds as follows.

If the input head exits to the right, $\mathcal{A}$ moves pebble 2 from m to $m + 1$. If $m + 1$ is not the start room then Z is supposed to be reading the first symbol of $[m + 1 * \sigma(m + 1, 1) * \sigma(m + 1, 2)]$. Pebble 2 is thus in the right place. Pebble 4 is then moved to room i where i is the length of $[m + 1$. Task 2 is used to compute the length of $[m + 1$. An adjustment of the finite control then completes the updating. If $m + 1$ is the start room, then there is no room $m + 1$ and pebble 2 is actually in room 1. So Z is supposed to be reading in $u_1 * u_2 * \cdots * u_g$. In this case, $\mathcal{A}$ picks up pebble 2. $\mathcal{A}$ then finds the lowest numbered goal room, that will be u_1. It then

drops pebble 5 in room u_1. Finally, $\mathcal{A}$ moves pebble 6 to room i, where i is the length of $u_1{}^*$.

If the input head exits $[m * \sigma(m, 1) * \sigma(m, 2)]$ to the left a similar, but slightly easier, routine updates the coding.

Case 2. *Input head reading left end marker.* An adjustment of the finite control of $\mathcal{A}$ updates the coded ID for any input head movement of Z.

Case 3. *Input head scanning initial one.* If the input head shifts left, an adjustment of the finite control of $\mathcal{A}$ updates the coded ID. If the input head shifts right, Z is then scanning in the block $[1 * \sigma(1, 1) * \sigma(1, 2)]$. So $\mathcal{A}$ moves pebble 2 to room 1 (the start room) to record this. Within this big block Z is scanning the leftmost symbol of the two-symbol subblock [1. So $\mathcal{A}$ moves pebble 4 to room 2 (the start room plus one) to record this fact. Finally, an adjustment of $\mathcal{A}$'s finite control completes the updating.

Case 4. *Input head reading in* $* u_j$. This is analogous to case 1.

Case 5. *Input head reading right-hand end marker.* The only possibility is a left shift. In this case Z would shift its input head so that it was scanning the first digit of $* u_g$. So $\mathcal{A}$ computes the highest numbered goal room. This is u_g. It then drops pebble 5 in room u_g to note that u_g is being scanned. It drops pebble 6 in room one (start room) to note that the first digit is being scanned. In order to find the highest numbered goal room, $\mathcal{A}$ simply goes through the rooms in reverse numerical order till it finds a goal room. $\mathcal{A}$ cannot literally go through the rooms in reverse numerical order; however, it can easily simulate this behavior by using task 1 (find predecessor).

This completes the description of $\mathcal{A}$'s updating routine and ends the proof of Theorem 1.

Nondeterministic Turing Machines

We now apply Theorem 1 to the problem of simulating nondeterministic tape bounded Turing machines by deterministic machines. In doing this, we will need the following result from [2].

Theorem 2. *For any finite alphabet Σ, with at least two elements, the following statements are equivalent.*

(1) *M_Σ is accepted by some deterministic Turing machine within storage* $\log_2 n$.

(2) *For any finite alphabet Γ, any set A, of strings over Γ and any function $L(n) \geqslant \log_2 n$, if A is accepted by a nondeterministic Turing machine within storage $L(n)$, then A is accepted by some deterministic Turing machine within storage $L(n)$.*

Combining Theorems 1 and 2 yields the main result of this paper.

THEOREM 3. *The following statements are equivalent.*

(1) *There is an* MRA $\mathcal{A}$, *such that for any* 2*-maze* $\mathcal{M}$, $\mathcal{A}$ *accepts* $\mathcal{M}$ *if and only if* $\mathcal{M}$ *is threadable.*

(2) *For any finite alphabet* Γ, *any set* A *of strings over* Γ *and any function* $L(n) \geqslant \log_2 n$, *if* A *is accepted by a nondeterministic Turing machine within storage* $L(n)$, *then* A *is accepted by a deterministic Turing machine within storage* $L(n)$.

COROLLARY. *If there is an* MRA $\mathcal{A}$ *which accepts exactly the threadable numbered* 2*-mazes, then every context-sensitive language is accepted by a deterministic linear-bound automaton.*

Proofs. To obtain the corollary from Theorem 3, first note that the context-sensitive languages are exactly those languages accepted by nondeterministic linear-bound automata [1]. Also, linear-bound automata are, by definition, Turing machines with a linear-bounded storage tape. So the corollary is obtained from Theorem 3 by taking $L(n) = n$.

Theorem 3 is obtained by combining Theorems 1 and 2. The result would be immediate except that Theorem 1 talks of M_Σ^2 (binary branching mazes) and Theorem 2 talks of M_Σ (arbitrary finite branching mazes). However, every nondeterministic Turing machine is equivalent to a machine in which there are at most two choices of moves in any situation. This fact, plus a careful reading of [2], will indicate that Theorem 2 could have been proven with M_Σ replaced by M_Σ^2. With this observation, Theorem 3 follows from Theorems 1 and 2. An alternate, more direct, method of obtaining Theorem 3 from Theorems 1 and 2 is to prove the following.

LEMMA 1. *For any finite alphabet* Σ *with at least two elements, the following statements are equivalent.*

(1) M_Σ *is accepted by some deterministic Turing machine within storage* $\log_2 n$.

(2) M_Σ^2 *is accepted by some deterministic Turing machine within storage* $\log_2 n$.

Theorem 3 is immediate from Lemma 1 and Theorems 1 and 2.

Proof of Lemma 1. Suppose (1) is true and Z_Σ is the deterministic Turing machine which accepts M_Σ in storage $\log_2 n$. By definition, $M_\Sigma^2 \subseteq M_\Sigma$. In fact, for any string w: $w \in M_\Sigma^2$ if and only if (i) w is a coding of a 2-maze and (ii) $w \in M_\Sigma$. So we can construct a deterministic Turing machine Z_2 to recognize M_Σ^2 as follows. For any input w, Z_2 first checks to see if (i) holds. This it can easily do in storage proportional to $\log_2 n$. Z_2 then mimicks Z_Σ to see if $w \in M_\Sigma$. Since Z_Σ operates in storage $\log_2 n$, Z_2 will also operate within storage $\log_2 n$.

Conversely, suppose (2) is true. Let Z_2 be the deterministic Turing machine which accepts $M_\Sigma{}^2$ within storage $\log_2 n$. We will construct a deterministic Turing machine Z_Σ to accept M_Σ within storage $\log_2 n$. Z_Σ will operate as follows. Given any coding of a maze over Σ. Z_Σ will recode this Σ-maze as an equivalent 2-maze. This is done in the usual way. Z_Σ then mimics Z_2 to see if the 2-maze is in $M_\Sigma{}^2$. If it is, then the original maze is threadable. So Z_2 accepts. The details are as follows.

Z_Σ can easily be designed to reject any string of symbols that is not a coding of a maze over Σ. So we may assume Z_Σ receives codings of Σ mazes as input. The input string

$$(*) \qquad s[x_1 * y_1{}^1 * y_2{}^1 * \cdots y^1_{n(1)}][x_2 * y_1{}^2 * y_2{}^2 * \cdots * y^2_{n(2)}]$$
$$\cdots [x_1 * y_1{}^l * \cdots y^l_{n(l)}]\, u_1 * u_2 * \cdots u_g$$

will be coded as an equivalent binary branching maze constructed as follows.

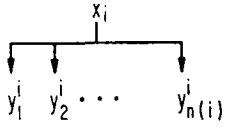

FIGURE 1

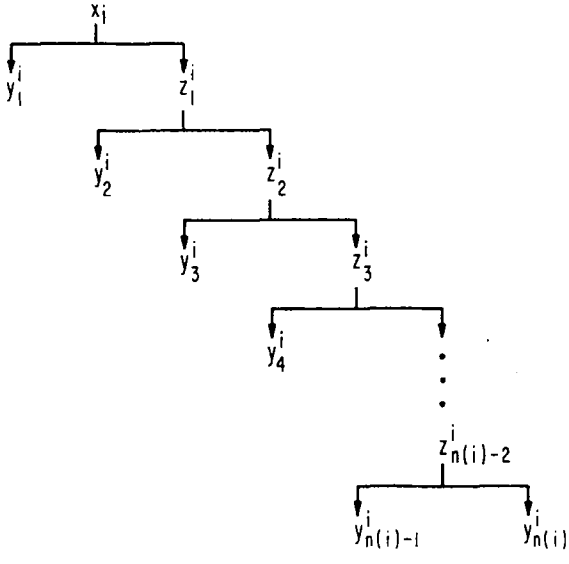

FIGURE 2

Each room x_i and the corridors leading from it may be diagramed as in Fig. 1. Replace each such node by the system of nodes in Fig. 2. The z_j^i are new rooms. This converts the original maze to a maze with binary branching. In order to get a 2-maze, we must label each node with a string in Σ^*. All of the nodes of the original maze already are labeled with strings in Σ^*. However, it will be more convenient to disregard this labeling and relabel all the nodes. The labeling is as follows. Let m be the number of symbols in Σ. After suitable identification, m-ary numerals are elements of Σ^*. The nodes of the 2-maze will be labeled by the m-ary numerals for the positive integers, 1, 2, 3,..., N, where N is the number of nodes. The rooms are numbered in the following order.

$$(**) \qquad x_1, z_1^1, z_2^1, \ldots, z_{n(1)-2}^1, x_2, z_1^2, \ldots, z_{n(2)-2}^2, x_3, z_1^3, z_2^3, \ldots, z_{n(3)-2}^3,$$

$$\cdots x_l, z_1^l, z_2^l, \ldots, z_{n(l)-2}^l.$$

Recall that the y_j^i's occur among the x_k's and so this labels all nodes. In what follows we will use x_i ambiguously to denote both a node in the original maze and the corresponding node in the associated 2-maze. y_j^i will similarly be used ambiguously. The intended meaning will be clear from the context. The 2-maze obtained this way will have fewer than $l \max n(i)$ nodes. Hence fewer than n^2 nodes. Hence the entire coding of the resulting 2-maze will be at most n^3 symbols long, where n is the length of the input (*). (It will actually have length about $l(\max n(i)) \log n$.)

Given input (*), Z_Σ operates as follows. Z_Σ constructs a coding of the associated 2-maze on one of its storage tapes, and then simulates Z_2 to determine whether the 2-maze is threadable. If it is then Z_Σ accepts the input (*). Except for the tape used to store the 2-maze, Z_Σ operates within storage $\log_2$ (length of the coding of the 2-maze) $\leqslant \log_2 n^3 \leqslant 3 \log_2 n$.

The machine Z_Σ cannot write down the entire coding of the 2-maze at once and still work within the alloted storage. However, all that is necessary in order to simulate Z_2 is that Z_Σ be able to compute one symbol at a time of the coded 2-maze and keep track of the symbols position in the code string. This it could do provided it could, within storage $\log_2 n$, generate the coded 2-maze from left to right, one symbol at a time. Z_Σ accomplishes this as follows.

Let $(i)_m$ denote the m-ary numeral for i. The coding of the 2-maze produced by Z_Σ is shown below.

$$(***) \quad (k)_m[(1)_m * q_1^1 * q_2^1][(2)_m * q_1^2 * q_2^2] \cdots [(N)_m * q_1^N * q_2^N] p_1 * p_2 * \cdots * p_g.$$

To produce this coding, Z_Σ first computes k. To do this Z_Σ scans the input (*) and compares s, symbol by symbol to each x_i. Since the comparison is symbol by symbol, Z_Σ need only remember two numbers, the position of the current symbol of s being checked and the position in (*) of the current symbol of x_i being checked. Each

number can be written in $\log_2 n$ storage or less. When Z_Σ finds the x_i which equals s, Z_Σ computes x_i's position in the list (**). This is k. x_i's position in (**) depends only on the number of x_j's and $y_k{}^j$'s which precede it in (*). More precisely, x_i's position in the list (**) is $(i-1)+\Sigma z(j)$ where the sum is over all $j<i$ and $z(j)$ is the number of new nodes $z_k{}^j$ added to transform node x_j to binary branding. $z(j)=n(j)-2$ if $n(j)>2$ and $z(j)=0$ if $n(j)=1$ or 2. $n(j)$ is the number of rooms reachable from x_j in one step, in the original maze (*). So x_i's position can be calculated by one sweep of the input head from x_i to the left-hand end of the input (*). So Z_Σ can compute k in the allotted storage.

Next, Z_Σ generates the pieces $[(i)_m * q_1{}^i * q_2{}^i]$ in order. To calculate $q_1{}^i$ and $q_2{}^i$, Z_Σ computes the i-th entry in the list (**) and then determines $q_1{}^i$ and $q_2{}^i$ from the input string as follows.

If the i-th entry is $z_k{}^j$ for some j and k, then $q_1{}^i = y_{k+1}^j$ and $q_2{}^i = z_{k+1}^j$ if there is a z_{k+1}^j in the list (**). If there is no z_{k+1}^j, then $q_2{}^i = y_{k+2}^i$. To determine the numeral coding y_{k+1}^i in the 2-maze (***), Z_Σ compares y_{k+1}^i to each x_t in the original input (*). When it finds that $x_t = y_{k+1}^i$, it then computes x_t's position in the list (**). This is the numeral for y_{k+1}^j and so gives the actual symbols generated as $q_1{}^i$.

If the i-th entry is one of the x_j's then $q_1{}^i = y_1{}^j$ and $q_2{}^i = z_1{}^j$. (If there is no $z_1{}^j$ then $q_2{}^i = y_2{}^j$. If there is no $y_2{}^j$ then $q_2{}^i$ does not appear in (***). (Similarly, if there is no $y_1{}^j$ then $q_1{}^i$ does not appear in (***).) The numerals for $y_1{}^j$ and $z_1{}^j$ are computed by the techniques given above.

Finally, the p_i are computed from u_i in (*). This is done in the same way as k was computed from s. This completes the description of how Z_Σ generates (***) and so completes the proof of Lemma 1.

Conclusions

Showing that deterministic and nondeterministic tape complexity classes are different is, as we have shown, equivalent to showing that no MRA can accept precisely the threadable mazes. We conjecture that no MRA can recognize the threadable mazes and see this as a way of showing that there is a difference between deterministic and nondeterministic tape complexity classes. However, all we have so far been able to prove is that: an MRA must have at least one pebble in order to recognize threadable mazes.

Acknowledgment

The author thanks Stephen A. Cook for many helpful comments and suggestions on this work.

REFERENCES

1. J. E. HOPCROFT AND J. D. ULLMAN, "Formal Languages and Their Relation to Automata," Addison–Wesley, Reading, Mass., 1969.
2. W. J. SAVITCH, Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.* **4** (1970), 177–192.
3. J. C. SHEPHERDSON AND H. E. STURGIS, "Computability of Recursive Functions," *J. Assoc. Comput. Mach.* **10** (1963), 217–255.